Authentication in PHP: Part 1

Building Secure User Systems with Auth.php

Authentication

What is Authentication?

Authentication vs Authorization

Authentication = "Who are you?"

- Log in with username/password
- Verify identity
- "Prove you are John Doe."

Authorization = "What can you do?"

- Check permissions
- Access control
- "John can edit posts, but not delete users"

Simple Example

```
User enters: username="john", password="secret123"
System checks: Does this match our records?
If yes: "Welcome John!" (Authenticated)
If no: "Invalid credentials" (Not authenticated)
```

Why Authentication Matters

Without Authentication

```php
// ❌ Anyone can access anything
if ($_GET['action'] == 'delete_user') {
    delete_user($_GET['id']); // Disaster!
}
```

With Authentication

```php
// ✅ Only logged-in users can perform actions
$auth = new Auth();
$user = $auth->get_current_user();

if ($user && $user['role'] == 'admin') {
    delete_user($_GET['id']);
} else {
    echo "Permission denied";
}
```

Authentication protects your application and users' data!

Basic Authentication Flow

1. Registration

```
User submits: username, password, email
System: Hash password → Store in database
Result: User account created
```

2. Login

```
User submits: username, password
System: Find user → Verify password hash
Result: Login successful → Create session
```

3. Protected Access

```
User requests protected page
System: Check session → Verify user
Result: Allow access or redirect to login
```

4. Logout

```
User clicks logout
System: Destroy session
Result: User logged out
```

Password Security Basics

❌ Never Store Plain Passwords

```
// WRONG - never do this!
$sql = "INSERT INTO users (username, password) VALUES (?, ?)";
$stmt->execute([$username, $password]); // Password visible!
```

✅ Always Hash Passwords

```php
// CORRECT - use password hashing
$password_hash = password_hash($password, PASSWORD_DEFAULT);
$sql = "INSERT INTO users (username, password_hash) VALUES (?, ?)";
$stmt->execute([$username, $password_hash]);
```

✅ Verify Hashed Passwords

```php
// Check login
if (password_verify($entered_password, $stored_hash)) {
    echo "Login successful!";
} else {
    echo "Invalid password!";
}
```

PHP's `password_hash()` and `password_verify()` are your friends!

What is a Hash?

- A **hash** is a fixed-length value generated from data using a **hash function**.
- It:
    - Represents the data uniquely (low collision chance)
    - Changes completely if the input changes slightly
    - Cannot be reversed to get the original data

Input String:

```
Hello World
```

Hashed String (SHA-256):

```
a591a6d40bf420404a011733cfb7b190
d62c65bf0bcda32b57b277d9ad9f146e
```

Key Properties

1. **Deterministic** – Same input → same hash
2. **Fast** – Quick to compute
3. **Fixed Length** – Always the same size output
4. **Avalanche Effect** – Small change → big hash change
5. **One-way** – Can’t reverse to get original

Everyday Uses

- 🔑 **Password storage** – Websites store hashes, not plain text passwords
- 🗂️ **File verification** – Ensure file integrity with hashes
- ✍️ **Digital signatures** – Verify message authenticity

Example: Password Hashing (PHP)

```php
<?php
$password = "MySecurePass123";
$hash = password_hash($password, PASSWORD_DEFAULT);
echo $hash;

if (password_verify("MySecurePass123", $hash)) {
    echo "Correct password!";
}
?>
```

Simple Authentication Example

basic_auth.php

- We use a session for authentication.
- We use a simple user database in JSON, but we use MySQL later in the course.

```php
<?php
session_start();

$users = [
    'john' => password_hash('secret123', PASSWORD_DEFAULT),
    'jane' => password_hash('mypassword', PASSWORD_DEFAULT)
];
```

Login process

- When the users give the correct username and password, we start a session.

```
if ($_POST['action'] == 'login') {
    $username = $_POST['username'];
    $password = $_POST['password'];

    if (isset($users[$username]) &&
        password_verify($password, $users[$username])) {

        $_SESSION['logged_in'] = true;
        $_SESSION['username'] = $username;
        echo "Welcome, $username!";
    } else {
        echo "Invalid credentials!";
    }
}
```

Check if logged in

```
if (isset($_SESSION['logged_in']) && $_SESSION['logged_in']) {
    echo "Hello, " . $_SESSION['username'] . "!";
    echo '<a href="?action=logout">Logout</a>';
} else {
    // Show login form
    ?>
    <form method="post">
        <input type="hidden" name="action" value="login">
        Username: <input type="text" name="username" required><br>
        Password: <input type="password" name="password" required><br>
        <button type="submit">Login</button>
    </form>
    <?php
}
```

Log out

```php
if ($_GET['action'] == 'logout') {
    session_destroy();
    header('Location: basic_auth.php');
}
?>
```

Auth Class Structure

Why Use Auth.php?

When you create a dedicated Auth class (Auth.php) instead of scattering authentication functions everywhere, you get:

1. Encapsulation of Related Logic
2. Reusability
3. Maintainability
4. Consistency
5. Testability
6. Extensibility

Comparison Table

Feature	Scattered Functions	Auth.php Class
Maintainability	Hard to update (many files)	Easy to update in one place
Consistency	Risk of different rules	Same rules everywhere
Reusability	Code repeated	Code reused via methods
Extensibility	Hard to add features	Easy to extend class
Testing	Hard to test	Easy to test independently

💡 **Key Point:**

Centralizing authentication in `Auth.php` means **one source of truth** for login, logout, and session management.

JsonDatabase.php

- This is an implementation of a simple JSON database.
- It supports simple CRUD operations.

```php
<?php

class JsonDatabase {
    private $file_path;
    private $data = [];

    public function __construct($file_path) {
        $this->file_path = $file_path;

        // Create data directory if it doesn't exist
        $dir = dirname($file_path);
        if (!is_dir($dir)) {
            mkdir($dir, 0755, true);
        }

        // Load data from file
```

- Loading and Saving JSON files

```
    // Load data from JSON file
    private function load_data() {
        if (file_exists($this->file_path)) {
            $json = file_get_contents($this->file_path);
            $this->data = json_decode($json, true) ?? [];
        } else {
            $this->data = [];
        }
    }

    // Save data to JSON file
    private function save_data() {
        $json = json_encode($this->data, JSON_PRETTY_PRINT);
        file_put_contents($this->file_path, $json);
    }
```

- CREAD: Adding a new record by making a new ID

```
    // Add new record
    public function add($record) {
        // Generate unique ID
        $id = $this->get_next_id();
        $record['id'] = $id;

        $this->data[] = $record;
        $this->save_data();

        return $record;
    }
```

- READ: Searching by ID and field value.

```php
// Get all records
public function read_data() {
    return $this->data;
}
// Find record by ID
public function find_by_id($id) {
    foreach ($this->data as $record) {
        if ($record['id'] == $id) {
            return $record;
        }
    }
    return null;
}
// Find record by field value
public function find_by_field($field, $value) {
    foreach ($this->data as $record) {
        if (isset($record[$field]) && $record[$field] === $value) {
            return $record;
        }
    }
    return null;
}
```

- UPDATE: Read by ID and update

```
    // Update record by ID
    public function update($id, $updates) {
        for ($i = 0; $i < count($this->data); $i++) {
            if ($this->data[$i]['id'] == $id) {
                $this->data[$i] = array_merge($this->data[$i], $updates);
                $this->save_data();
                return $this->data[$i];
            }
        }
        return null;
    }
```

- DELETE: Search by ID and delete

```
    // Delete record by ID
    public function delete($id) {
        for ($i = 0; $i < count($this->data); $i++) {
            if ($this->data[$i]['id'] == $id) {
                $deleted = $this->data[$i];
                array_splice($this->data, $i, 1);
                $this->save_data();
                return $deleted;
            }
        }
        return null;
    }
```

- It supports other utility functions.

```
    // Get next available ID
    private function get_next_id() {
        $max_id = 0;
        foreach ($this->data as $record) {
            if (isset($record['id']) && $record['id'] > $max_id) {
                $max_id = $record['id'];
            }
        }
        return $max_id + 1;
    }

    // Count records
    public function count() {
        return count($this->data);
    }

    // Clear all data
    public function clear() {
        $this->data = [];
        $this->save_data();
    }
}
```

Auth.php

This class has all the functions for authentication.

- Registration with username and hashed password
- Login
- Validation
- Login attempt limitation
- User activation/deactivation
- User Profile
- Password change
- Find Users
- User information management

Property and constructor

- We use JSON for database storage for education; we will replace it with MySQL later in this class.

```php
class Auth {
    private $users_db;

    public function __construct($users_file = 'data/users.json') {
        $this->users_db = new JsonDatabase($users_file);
    }
```

Registration

- Register method to hash users' passwords.

```
    public function register($username, $password, $email = null) {
        // Check if user exists
        if ($this->find_user_by_username($username)) {
            throw new Exception('Username already exists');
        }
        // Validate password
        if (strlen($password) < 6) { throw new Exception('Password must be at least 6 characters'); }
        // Hash password and create user
        $password_hash = password_hash($password, PASSWORD_DEFAULT);
        $user_data = [
            'username' => $username,
            'password_hash' => $password_hash,
            'email' => $email,
            'created_at' => date('c'),
            'is_active' => true,
            'login_attempts' => 0
        ];
        return $this->users_db->add($user_data);
    }
```

Login

- We can implement the login method using the utility functions.

```
    public function login($username, $password) {
        $user = $this->find_user_by_username($username);

        if (!$user || !password_verify($password, $user['password_hash'])) {
            throw new Exception('Invalid credentials');
        }

        return $user;
    }
}
```

Validation

- We need to validate the password strength.

```
    // Validate password strength
    public function validate_password_strength($password) {
        $errors = [];
        if (strlen($password) < 6) { $errors[] = "at least 6 characters"; }
        if (!preg_match('/[A-Z]/', $password)) { $errors[] = "at least one uppercase letter"; }
        if (!preg_match('/[a-z]/', $password)) { $errors[] = "at least one lowercase letter"; }
        if (!preg_match('/[0-9]/', $password)) { $errors[] = "at least one number"; }

        if (!empty($errors)) {
            throw new Exception("Password must have " . implode(', ', $errors));
        }

        return true;
    }
```

Login attempts limitation

- Check if user is rate limited

```
    private function is_rate_limited($user) {
        if (($user['login_attempts'] ?? 0) >= 5) {
            $last_attempt = strtotime($user['last_attempt'] ?? '');
            $now = time();
            $time_diff = $now - $last_attempt;

            // Rate limit for 15 minutes (900 seconds)
            return $time_diff < 900;
        }

        return false;
    }
```

- Record failed login attempt

```
    private function record_failed_attempt($user_id) {
        $user = $this->users_db->find_by_id($user_id);

        $this->users_db->update($user_id, [
            'login_attempts' => ($user['login_attempts'] ?? 0) + 1,
            'last_attempt' => date('c')
        ]);
    }
```

User activation/deactivation

```
    // Deactivate user
    public function deactivate_user($user_id) {
        return $this->users_db->update($user_id, [
            'is_active' => false,
            'deactivated_at' => date('c')
        ]);
    }

    // Activate user
    public function activate_user($user_id) {
        return $this->users_db->update($user_id, [
            'is_active' => true,
            'activated_at' => date('c')
        ]);
    }
```

User Profile

```
    // Update user profile
    public function update_profile($user_id, $updates) {
        // Only allow safe fields to be updated
        $allowed_fields = ['email'];
        $safe_updates = [];

        foreach ($allowed_fields as $field) {
            if (isset($updates[$field])) {
                // Special validation for email
                if ($field === 'email') {
                    if (!filter_var($updates[$field], FILTER_VALIDATE_EMAIL)) {
                        throw new Exception('Invalid email format');
                    }

                    // Check email uniqueness
                    $existing_email = $this->find_user_by_email($updates[$field]);
                    if ($existing_email && $existing_email['id'] != $user_id) {
                        throw new Exception('Email already registered to another user');
                    }
                }

                $safe_updates[$field] = $updates[$field];
            }
        }

        $safe_updates['updated_at'] = date('c');

        return $this->users_db->update($user_id, $safe_updates);
    }
```

Password change

```
    // Change user password
    public function change_password($user_id, $old_password, $new_password) {
        $user = $this->find_user_by_id($user_id);

        if (!$user) {
            throw new Exception('User not found');
        }

        // Verify old password
        if (!password_verify($old_password, $user['password_hash'])) {
            throw new Exception('Current password is incorrect');
        }

        // Validate new password strength
        $this->validate_password_strength($new_password);

        // Hash new password
        $new_hash = password_hash($new_password, PASSWORD_DEFAULT);

        // Update password
        $this->users_db->update($user_id, [
            'password_hash' => $new_hash,
            'password_changed_at' => date('c')
        ]);

        return true;
    }
```

Find users

- Search functions

```
    // Find user by username
    public function find_user_by_username($username) {
        return $this->users_db->find_by_field('username', $username);
    }

    // Find user by email
    public function find_user_by_email($email) {
        return $this->users_db->find_by_field('email', $email);
    }

    // Find user by ID
    public function find_user_by_id($id) {
        return $this->users_db->find_by_id($id);
    }

    ...
```

User Information Management

- Get all users (for admin purposes)

```
    public function get_all_users($include_inactive = false) {
        $users = $this->users_db->read_data();

        $filtered_users = [];
        foreach ($users as $user) {
            // Filter by active status
            if (!$include_inactive && !($user['is_active'] ?? true)) {
                continue;
            }
            // Remove sensitive data
            $safe_user = $user;
            unset($safe_user['password_hash']);
            $filtered_users[] = $safe_user;
        }
        return $filtered_users;
    }
```

- Get user statistics

```php
public function get_user_stats() {
    $users = $this->users_db->read_data();

    $stats = [
        'total_users' => count($users),
        'active_users' => 0,
        'inactive_users' => 0,
        'recent_registrations' => 0, // Last 7 days
        'recent_logins' => 0 // Last 24 hours
    ];

    $now = time();
    $week_ago = $now - (7 * 24 * 60 * 60);
    $day_ago = $now - (24 * 60 * 60);

    foreach ($users as $user) {
        // Count active/inactive
        if ($user['is_active'] ?? true) {
            $stats['active_users']++;
        } else {
            $stats['inactive_users']++;
        }
        // Count recent registrations
        if (strtotime($user['created_at']) > $week_ago) {
            $stats['recent_registrations']++;
        }
        // Count recent logins
        if (isset($user['last_login']) && strtotime($user['last_login']) > $day_ago) {
            $stats['recent_logins']++;
        }
    }
    return $stats;
}
```